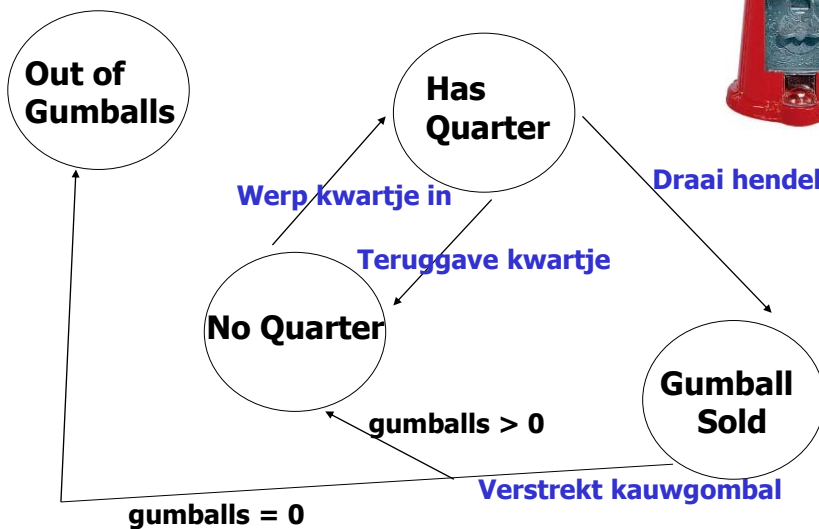
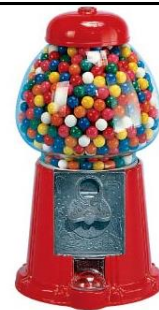


State Pattern – gedrag van objecten

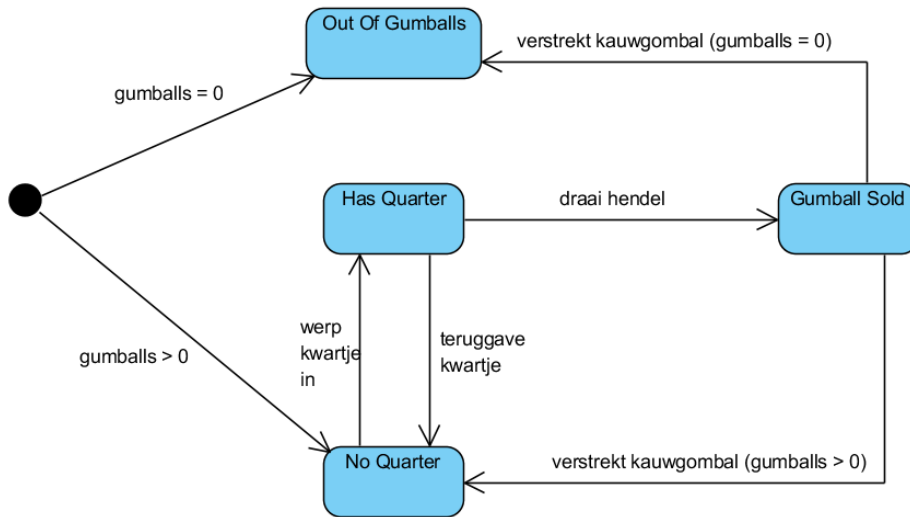
1

Voorbeeld: automaat in kauwgomballen



Voorbeeld: automaten in kauwgomballen In Netbeans: State Machine Diagram:

OOAD

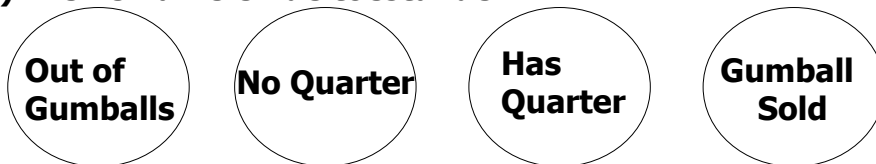


3

Van toestandsdiagram naar code.

OOAD

1) We verzamelen de toestanden



2) We maken een instantievariabele om de huidige toestand te bevatten, en definiëren we waarden voor iedere toestand:

```

GumballMachine
-OUT_OF_GUMBALLS : int = 0
-NO_QUARTER : int = 1
-HAS_QUARTER : int = 2
-SOLD : int = 3
-currentState : int = GumballMachine.OUT_OF_GUMBALLS

```

```

private final static int OUT_OF_GUMBALLS = 0;
private final static int NO_QUARTER = 1;
private final static int HAS_QUARTER = 2;
private final static int SOLD = 3;

```

```

private int currentState = OUT_OF_GUMBALLS;

```

Van toestandsdiagram naar code.



3) We verzamelen alle acties

Werp kwartje in

Draai hendel

Teruggave kwartje

Verstrekt kauwgombal (is een interne actie die de machine zelf activeert)

4) Voor iedere actie maken we een methode.

```
GumballMachine
-OUT_OF_GUMBALLS : int = 0
-NO_QUARTER : int = 1
-HAS_QUARTER : int = 2
-SOLD : int = 3
-currentState : int = slides.GumballMachine.OUT_OF_GUMBALLS
+GumballMachine(numberGumballs : int)
+insertQuarter() : String
+ejectQuarter() : String
+turnCrank() : String
+refill(count : int) : void
+toString() : String
#dispense() : String
```

insertQuarter: wanneer een kwartje is ingeworpen.

ejectQuarter: als de klant zijn kwartje terug wil hebben.

turnCrank: de klant probeert de hendel te draaien.

dispense: wordt aangeroepen om een kauwgombol te verstrekken.

Bv. Methode insertQuarter



```
public String insertQuarter() {
    switch(currentState)
    {
        case HAS_QUARTER :
            return "You can't insert another quarter";

        case NO_QUARTER :
            currentState = HAS_QUARTER;
            return "You inserted a quarter";

        case OUT_OF_GUMBALLS:
            return "You can't insert a quarter, the machine is sold out";

        case SOLD:
            return "Please wait, we're already giving you a gumball";
    }
}
```

public class StartUp{

**public static void main(String[] args) {
 GumballMachine gumballMachine =
 new GumballMachine(5);**

System.out.println(gumballMachine);

**System.out.println(
 gumballMachine.insertQuarter());
 System.out.println(
 gumballMachine.turnCrank());**

System.out.println(gumballMachine);

**System.out.println(
 gumballMachine.insertQuarter());
 System.out.println(gumballMachine.ejectQuarter());
 System.out.println(gumballMachine.turnCrank());**

System.out.println(gumballMachine); ...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
Quarter returned
You can't turn
You need to pay first

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model
Inventory: 4 gumballs
Machine is waiting for quarter

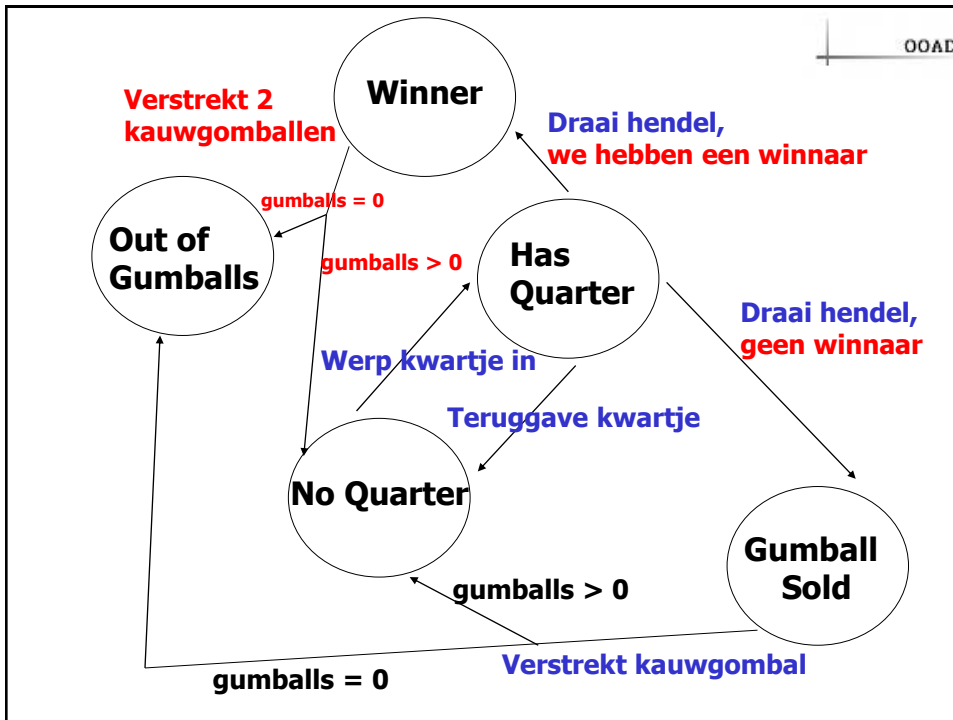
OOAD



Uitbreiding

In 10% van de gevallen krijgt de klant als hij aan de hendel draait, twee kauwgomballen in plaats van één.

OOAD



Dat je de kauwgomballenautomaat met een goeddoordachte methode hebt geschreven, betekent nog niet dat deze eenvoudig uitgebreid kan worden. Als je je code nog eens bekijkt en bedenkt wat je allemaal moet veranderen, nou ...

```

private final static int OUT_OF_GUMBALLS = 0;
private final static int NO_QUARTER = 1;
private final static int HAS_QUARTER = 2;
private final static int SOLD = 3;
  
```

```
public String insertQuarter() {...}
```

```
public String ejectQuarter() {...}
```

```
public String turnCrank() {...}
```

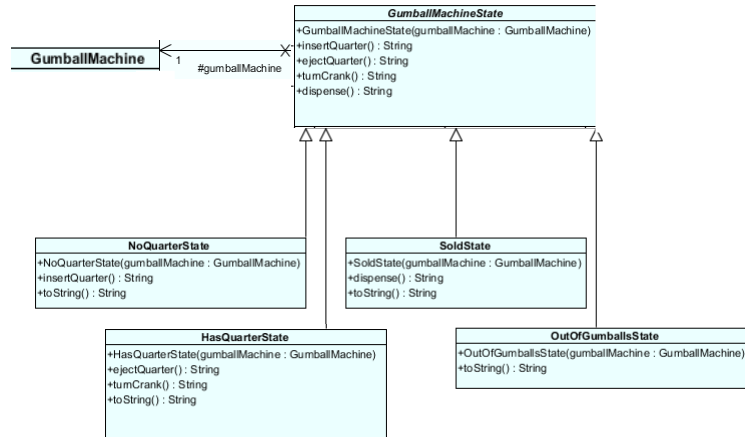
```
protected String dispense() {...}
```

Je moet in iedere methode een nieuwe conditioneel statement voor het afhandelen van de toestand WINNER aanbrengen (veel code wijzigen)

turnCrank: wordt onoverzichtelijk, omdat je code moet toevoegen om vast te stellen of je een WINNER hebt. Daarna schakel je door naar de toestand WINNER of de toestand SOLD.

Applicatie beschrijven (zonder uitbreiding):

Dit is de **abstracte klasse** voor al onze toestanden.



We nemen iedere toestand in ons ontwerp en schermen deze af in een klasse die erft van de toestandsklasse.

11

abstract class GumballMachineState {



protected final GumballMachine gumballMachine;

```

public GumballMachineState(
    GumballMachine gumballMachine) {
    this.gumballMachine = gumballMachine;
}
    
```

Zeg tegen de klant, 'Je kan geen kwartje inwerpen

```

public String insertQuarter()
{
    return "You can't insert a quarter";
}
    
```

Zeg tegen de klant, 'Je hebt geen kwartje ingeworpen

```

public String ejectQuarter()
{
    return "You haven't inserted a quarter";
}
    
```

**Zeg tegen de klant ` Je draait
maar je moet eerst betalen.**

```

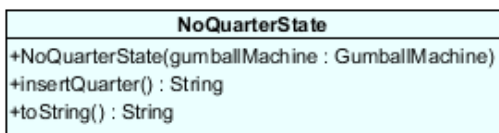
public String turnCrank()
{
    return "You turn but you need to pay first ";
}

public String dispense()
{
    return "You need to pay first";
}
}
    
```

**Zeg tegen de klant ` Je moet
eerst betalen.**



Ga naar HasQuarterState



```

class NoQuarterState extends GumballMachineState
{
    public NoQuarterState(
        GumballMachine gumballMachine)
    {
        super(gumballMachine);
    }

    public String insertQuarter() {
        gumballMachine.toState(
            new HasQuarterState(gumballMachine));
        return "You inserted a quarter";
    }

    public String toString() {
        return "waiting for quarter";
    }
}

```

De kauwgomballenautomaat ombouwen

OOAD

```

private final static int OUT_OF_GUMBALLS = 0;
private final static int NO_QUARTER = 1;
private final static int HAS_QUARTER = 2;
private final static int SOLD = 3;
private int state = OUT_OF_GUMBALLS;

```

```

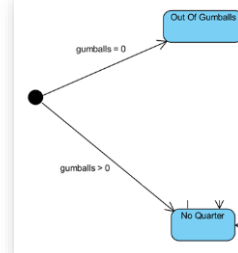
public class GumballMachine {

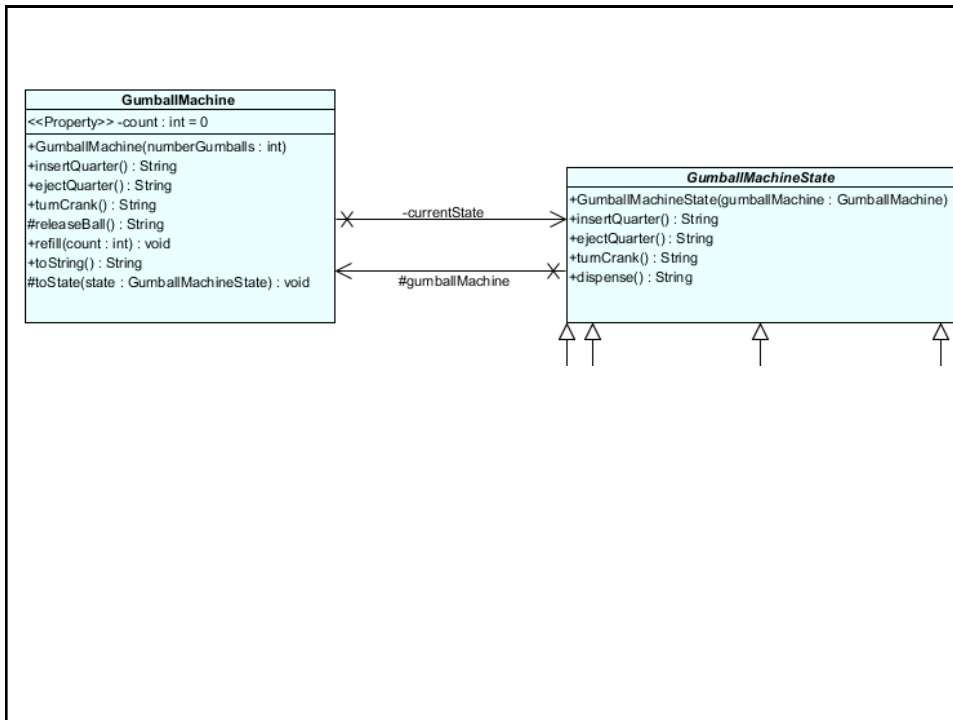
    private GumballMachineState currentState;

    public GumballMachine(int numberGumballs) {
        this.count = numberGumballs;
        if (numberGumballs > 0)
            toState(new NoQuarterState(this));
        else
            toState(new OutOfGumballsState(this));
    }

    protected void toState(GumballMachineState state) {
        currentState = state;
    } ...
}

```



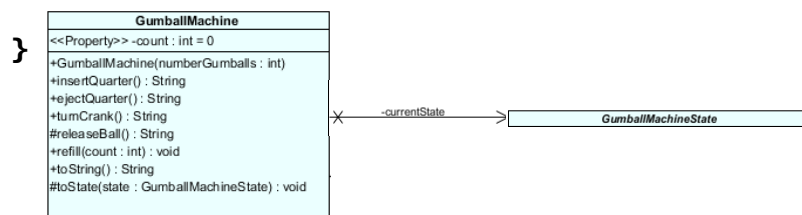


```
public class GumballMachine {
```

```
    //De instantievariabele voor de toestand:
    private GumballMachineState currentState;
```

```
    //De instantievariabele bevat het aantal kauwgomballen
    private int count = 0;
```

```
    public GumballMachine(int numberGumballs) {
        this.count = numberGumballs;
        if (numberGumballs > 0)
            toState(new NoQuarterState(this));
        else
            toState(new OutOfGumballsState(this));
    }
```



/*Nu de acties. Deze kun je nu ZEER EENVOUDIG implementeren. We delegeren dat aan de huidige toestand. */

```
public String insertQuarter() {
    return currentState.insertQuarter();
}

public String ejectQuarter() {
    return currentState.ejectQuarter();
}

public String turnCrank() {
    String msg1 = currentState.turnCrank();
    String msg2 = currentState.dispense();
    return String.format(
        "%s\n%s", msg1, msg2);
}
```

/*We hebben geen actiemethode voor dispense() in GumballMachine nodig, omdat dit een interne actie betreft. */

/*De machine ondersteunt de hulpmethode releaseBall() die de bal vrijgeeft en de instantievariabele count verlaagt.*/

```
protected String releaseBall() {
    if (count != 0) {
        count = count - 1;
    }
    return "A gumball comes rolling out the slot...";
}
```

//De machine vullen met kauwgommen. De toestand // wordt op 'No Quarter' gezet.

```
public void refill(int count) {
    if (count > 0) {
        this.count = count;
        toState(new NoQuarterState(this));
    }
}
```

```

    /*via deze methode kan een toestandsobject de machine in
    een andere toestand brengen.*/
    protected void toState(GumballMachineState state) {
        currentState = state;
    }

    public int getCount() { return count; }

    public String toString() {
        StringBuilder result = new StringBuilder();
        result.append("\nMighty Gumball, Inc.");
        result.append("\nJava-enabled Standing Gumball Model");
        result.append("\nInventory: " + count + " gumball");
        if (count != 1) {
            result.append("s");
        }
        result.append("\n");
        result.append("Machine is " + currentState + "\n");
        return result.toString();
    }
}

```

Meer toestanden implementeren

```

class HasQuarterState extends GumballMachineState
{
    public HasQuarterState(GumballMachine gumballMachine) {
        super(gumballMachine);
    }

    public String ejectQuarter() {
        gumballMachine.toState(new NoQuarterState(gumballMachine));
        return "Quarter returned";
    }

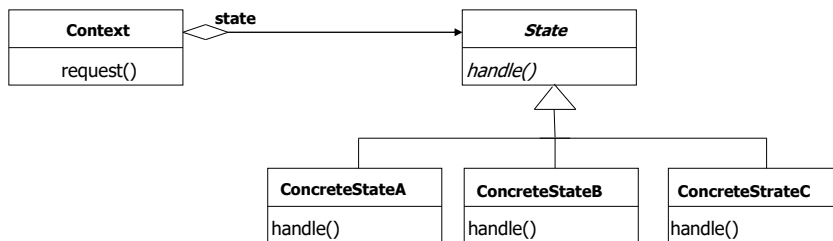
    public String turnCrank() {
        gumballMachine.toState(new SoldState(gumballMachine));
        return "You turned...";
    }

    public String toString() {
        return "waiting for turn of crank";
    }
}

```

State pattern


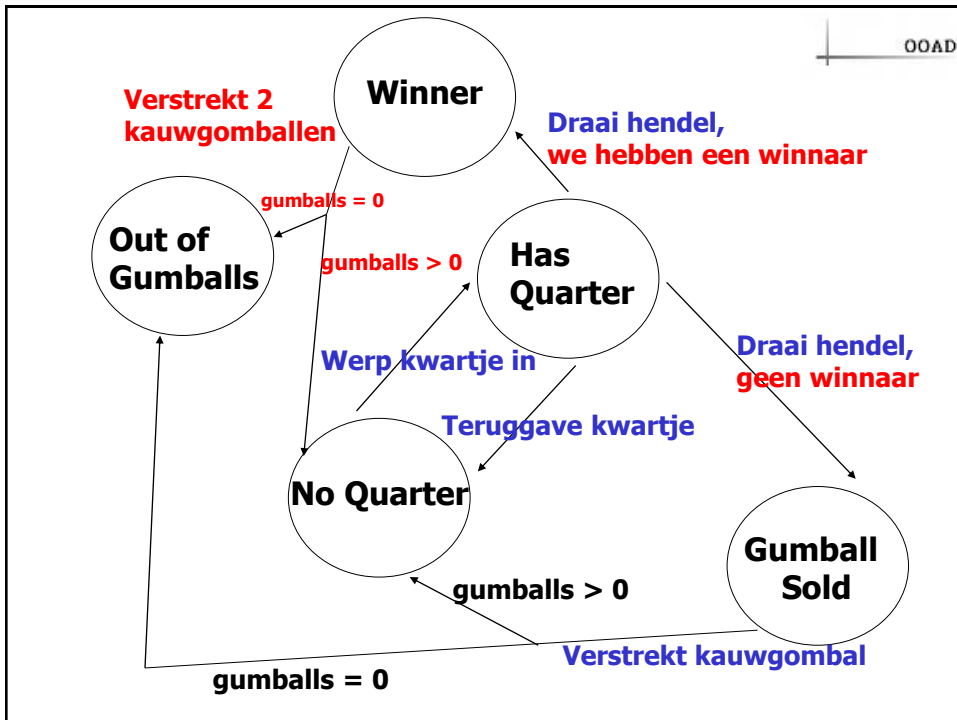
Het State Pattern maakt het voor een object mogelijk zijn gedrag te veranderen wanneer zijn interne toestand verandert. Het object lijkt van klasse te veranderen.



UITBREIDING:

In 10% van de gevallen krijgt de klant als hij aan de hendel draait, twee kauwgomballen in plaats van één.

Teken een toestandsdiagram voor een kauwgomballenautomaat die het een-op-tienspel afhandelt. In dit spel worden 10% van de keren dat je toestand **SOLD** wordt bereikt, twee kauwgomballen uitgegeven en niet één.

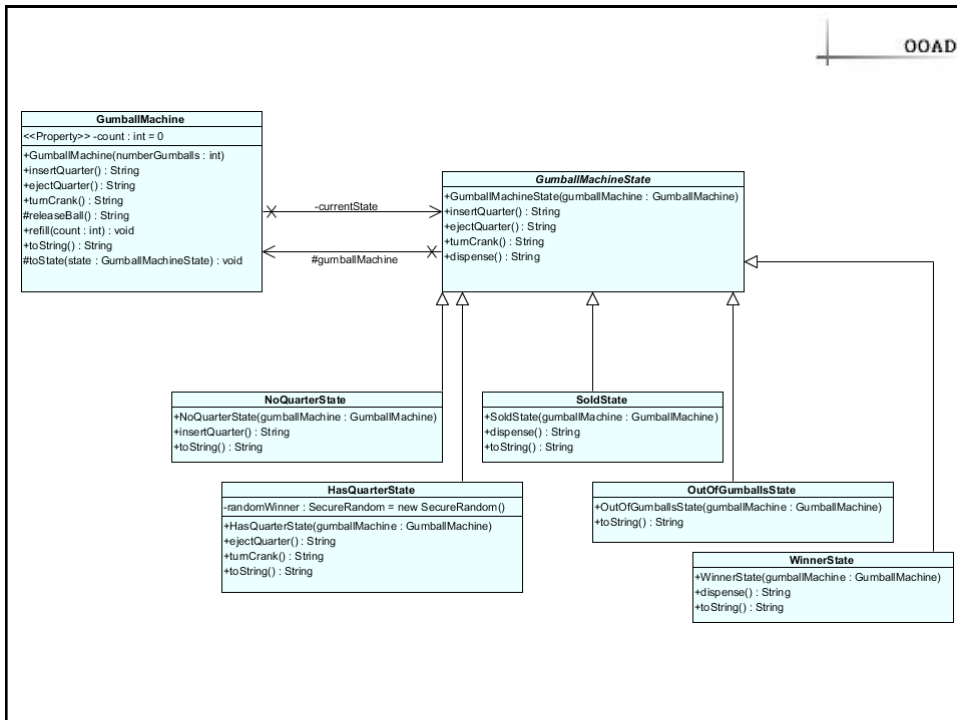


OOAD

UITBREIDING:
In 10% van de gevallen krijgt de klant als hij aan de hendel draait, twee kauwgomballen in plaats van één.

Implementeer het spel.

26



```

class WinnerState extends GumballMachineState {
    public WinnerState(GumballMachine gumballMachine) {
        super(gumballMachine);
    }
    public String dispense() {
        String msg =
            "YOU'RE A WINNER! You get two gumballs for your quarter\n";
        msg += gumballMachine.releaseBall();
        if (gumballMachine.getCount() == 0) {
            msg += "\nOops, out of gumballs!";
            gumballMachine.toState(
                new OutOfGumballsState(gumballMachine));
        } else {
            msg += gumballMachine.releaseBall();
            if (gumballMachine.getCount() > 0) {
                gumballMachine.toState(
                    new NoQuarterState(gumballMachine));
            } else {
                gumballMachine.toState(
                    new OutOfGumballsState(gumballMachine));
            }
        }
        return msg;
    }
} ...
  
```

WinnerState

+WinnerState(gumballMachine : GumballMachine)
 +dispense() : String
 +toString() : String

```
import java.security.SecureRandom;
```

```
class HasQuarterState  
    extends GumballMachineState
```

```
{
```

```
    private SecureRandom randomWinner = new SecureRandom();
```

```
    public String turnCrank() {  
        int winner = randomWinner.nextInt(10);  
        if ((winner == 0) &&  
            (gumballMachine.getCount() > 1)) {  
            gumballMachine.toState(  
                new WinnerState(gumballMachine));  
        } else {  
            gumballMachine.toState(  
                new SoldState(gumballMachine));  
        }  
        return "You turned...";  
    }
```

```
}
```

```
...
```

| HasQuarterState |
|---|
| -randomWinner : SecureRandom = new SecureRandom() |
| +HasQuarterState(gumballMachine : GumballMachine) |
| +rejectQuarter() : String |
| +turnCrank() : String |
| +toString() : String |